Math 172—Spring 2010—Haiman
Notes on ordinary generating functions

1. How do we count with generating functions?

Many enumeration problems which are not so easy to handle by elementary means can be solved using generating functions. In these notes I'll describe the general set-up in which ordinary generating functions apply. Exponential generating functions are useful in a somewhat different set-up, which we will discuss later.

Our basic problem will be to count objects of a given size, or *weight*, n. The specific meaning of the weight will vary with the context, but here are some typical possibilities: n might be the length of a word, or the size of a subset or multisubset of a given set, or we might be counting compositions of n, or partitions of n. We can also have situations in which there is more than one relevant weight. For example in counting partitions $\lambda = (\lambda_1, \dots, \lambda_k)$, we may want to keep track of both the size of the partition $n = w(\lambda) = \lambda_1 + \lambda_2 + \cdots + \lambda_k$, and the number of parts $l(\lambda) = k$, to arrive at the number $p(n,k)$ of partitions of n with k parts.

What often happens is that if we don't fix the weight in advance, but rather try to count objects of all weights simultaneously, it allows us extra freedom to make independent choices and thereby simplify the problem. But we have to do this in a way that keeps track of the weights, so we can solve our original problem. We will also have to deal with the fact that the set of all objects of all possible weights is typically infinite. Both of these difficulties can be handled by generalizing our usual idea of the number of elements of a finite set to a *weight enumerator* for elements of a possibly infinite set with weights.

Definition Let $\mathcal{S}$ be a set equipped with a function $w\colon \mathcal{S} \to \mathbb{N}$, called the *weight function*. Assume that for each n, the subset $\{s \in \mathcal{S} : w(s) = n\}$ is finite, and let $s_n = |\{s \in \mathcal{S} : w(s) = n\}|$ be its number of elements (the whole set $\mathcal{S}$ is allowed to be infinite). The *weight enumerator* for $\mathcal{S}$ is the generating function

$$S(x) = \sum_{n=0}^{\infty} s_n x^n = \sum_{s \in \mathcal{S}} x^{w(s)}.$$

Note that if $\mathcal{S}$ is finite, then $S(1)$ is just the number of elements of $\mathcal{S}$. In general, $S(x)$ should be thought of as a kind of weighted count of the elements of $\mathcal{S}$, where an element of weight n counts as x^n instead of 1.

The sum principle for weight enumerators is just like that for elementary counting, and follows immediately from the definition:

Sum Principle. If $\mathcal{S}$ is the union $\mathcal{S} = \mathcal{A} \cup \mathcal{B}$ of two disjoint subsets $\mathcal{A} \cap \mathcal{B} = \emptyset$, then

$$S(x) = A(x) + B(x),$$

where $S(x)$, $A(x)$, $B(x)$ denote the weight enumerators for $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{B}$ respectively (taking the weight functions on $\mathcal{A}$ and $\mathcal{B}$ to be the restrictions of the given one on $\mathcal{S}$).

There is also an analog for weight enumerators of the product principle from elementary counting. For this, however, we have to be careful about how weights on the elements of the product are related to those on the components: the weights are required to be *additive*.

Product Principle. If $\mathcal{S}$ is a Cartesian product $\mathcal{S} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_k$ of weighted sets, *and if the weights satisfy* $w(s) = w(a_1) + \cdots + w(a_k)$ *for* $s = (a_1, \dots, a_k)$, then

$$S(x) = A_1(x) A_2(x) \cdots A_k(x)$$

where $S(x)$, $A_i(x)$ denote the weight enumerators for $\mathcal{S}$ and the sets $\mathcal{A}_i$.

To see why the product principle works, observe that for two sets $\mathcal{S} = \mathcal{A} \times \mathcal{B}$, we have

$$S(x) = \sum_{(a,b)\in\mathcal{S}} x^{w((a,b))} = \sum_{(a,b)\in\mathcal{S}} x^{w(a)+w(b)} = \sum_{(a,b)\in\mathcal{S}} x^{w(a)}x^{w(b)} = \sum_{a\in\mathcal{A}} x^{w(a)} \sum_{b\in\mathcal{B}} x^{w(b)} = A(x)B(x).$$

The general case of a product of k sets follows from the two-set case by induction on k.

If our set $\mathcal{S}$ is equipped with more than one weight function, we will have a corresponding weight enumerator which is a generating function in more than one variable, for example, if we have two weight functions w and z, we would have a weight enumerator

$$S(x,y) = \sum_{s\in\mathcal{S}} x^{w(s)}y^{z(s)} = \sum_{n=0}^{\infty}\sum_{k=0}^{\infty} s_{n,k}x^n y^k,$$

where $s_{n,k}$ is the number of elements $s \in S$ such that $w(s) = n$ and $z(s) = k$.

2. Words, compositions.

In many instances, typified by the examples in this section, we can assemble an object from smaller pieces by first deciding on the number of pieces to use, say j, and then choosing the j pieces independently, in a definite order. This leads to situations in which we use the product principle for each j, and then sum over all j, leading to a geometric series involving a previously known generating function.

As a first example, let us determine the number g_n of words w of length n in the symbols $\{0, 1\}$ which do not contain consecutive 1's. Before considering the generating function, we might observe that g_n satisfies the same recurrence as the Fibonacci numbers. To see this, note that the word w must either start with a 0 or a 1. If it starts with a 0, the rest of the word can be any word of length $n-1$ without consecutive 1's, giving g_{n-1} choices. Provided that $n \geq 2$, if our word starts with 1, then it must in fact start with 10. This can be followed by anything, giving g_{n-2} choices. Hence

$$g_n = g_{n-1} + g_{n-2}, \quad (n \geq 2).$$

Since we also clearly have $g_0 = 1$ and $g_1 = 2$, we can deduce that g_n is equal to the Fibonacci number F_{n+1}. We could in turn obtain the generating function by using the recurrence, as explained in your textbook. Instead we'll take a different approach.

As a little thought will convince you, any word without consecutive 1's can be built up in a unique way by concatenating a sequence of subwords of the forms

$$0 \quad \text{or} \quad 10,$$

followed by an optional 1 at the end. Assigning each word a weight equal to its length, their weight enumerator is $G(x) = \sum_{n=0}^{\infty} g_n x^n$. Let us try to evaluate this directly.

Suppose our word is built from j subwords 0 or 10. We can choose each of these independently and multiply together the weight enumerators for each of the j choices. The weights are additive, since the total length of the word we build up will be the sum of the lengths of the subwords we build it from.

Now for each choice of a subword 0 or 10, we have a corresponding weight enumerator $(x+x^2)$, so we get $(x+x^2)^j$ as the generating function enumerating all concatentations of j of these subwords. Summing this over all j, we get

$$\sum_{j=0}^{\infty}(x+x^2)^j = \frac{1}{1-(x+x^2)}$$

as the generating function for all words (including the empty word) built by concatenating any sequence of any number of subwords 0 or 10. Finally, we can choose to add a 1 at the end or not, either contributing nothing or one extra to the weight, so we should multiply the above generating function by $1 + x$, obtaining

$$G(x) = \frac{1+x}{1-x-x^2}.$$

Note, by the way, that $xG(x)+1 = 1/(1-x-x^2)$. The coefficient of x^n in $xG(x)+1$ is $g_{n-1} = F_n$ for $n \geq 1$, and is $1 = F_0$ for $n = 0$, so the Fibonacci number generating function is given by

$$F(x) = \sum_{n=0}^{\infty} F_n x^n = \frac{1}{1-x-x^2},$$

in agreement with the formula you would obtain by using the recurrence (see Bona, Chapter 8, Ex. 4).

What we have done in this example is an instance of a more general method, which we may formalize as a separate enumeration principle for generating functions.

Sequence principle. Let $\mathcal{A}$ be a set with weighted elements and generating function $A(x)$. Assume there are no elements of weight zero, so $A(0) = 0$. Then the generating function for ordered sequences $\alpha_1, \alpha_2, \ldots, \alpha_j$ of of elements of $\mathcal{A}$ (of any length, including the empty sequence), with weight defined to be the sum of the weights of the elements α_i, is given by

$$\frac{1}{1-A(x)}.$$

Some caution is due, when using the sequence principle, to be sure the set $\mathcal{A}$ has no elements of weight zero. This is essential for two reasons. First of all, the expression $1/(1-A(x))$ makes no sense as a formal series if A has non-zero constant term. Secondly, it makes no sense to enumerate all sequences if there is an element α of weight zero in $\mathcal{A}$, since we could pad any sequence with an unlimited number of copies of α without changing its weight, giving infinitely many sequences of the same weight. (These two difficulties are related to each other.)

Here are some more examples.

Example. Let us find the generating function for all words from the alphabet $[k] = \{1, \ldots, k\}$. In this case $\mathcal{A}$ is just $[k]$, and each of its k elements has weight 1, so $A(x) = kx$. Hence the generating function for all words is

$$\frac{1}{1-kx} = \sum_n k^n x^n,$$

which shows that there are k^n words with n letters from $[k]$. Of course we knew this already. This example was just a warm-up.

Example. A *composition* of n is an expression

$$n = \lambda_1 + \lambda_2 + \cdots + \lambda_k$$

of n as a sum of positive integers. In a composition of n, as a opposed to a partition of n, the order of the parts matters. Thus $2+3+2$ and $3+2+2$, for instance, count as two different compositions of 7. Let $m(n,k)$ denote the number of compositions of n with k parts. We will find the generating function $M(n,k) = \sum_{n,k} m(n,k)x^n y^k$, and use it to solve for $m(n,k)$. In this case, we will use the sequence principle for a two-variable generating function, which works just like the one-variable case. Note that we are not fixing n or k in advance, so a composition is just an arbitrary sequence of positive integers. The set $\mathcal{A}$ here is therefore the set of all positive integers. We have assigned each composition two weights. One is its length, which is the exponent of y, while the other is

its size, which is the exponent of x in our generating function. Each integer i in the composition contributes one to the length and i to the size, so is counted by the monomial $x^i y$ in the generating function for $\mathcal{A}$. Summing over all $i \geq 1$ this gives $A(x,y) = yx/(1-x)$, and hence

$$M(x,y) = \frac{1}{1 - yx/(1-x)} = \frac{1-x}{1-x-xy}.$$

To solve for $m(n,k)$ it is convenient to rewrite this as

$$\begin{aligned} M(x,y) = 1 + xy\frac{1}{1-x(1+y)} &= 1 + xy\sum_{j=0}^{\infty} x^j(1+y)^j \\ &= 1 + xy\sum_j x^j \sum_i \binom{j}{i} y^i. \end{aligned}$$

Extracting the coefficient of $x^n y^k$, we see that this monomial occurs only in the term with $j = n-1$, $i = k-1$, so

$$m(n,k) = \binom{n-1}{k-1}.$$

Stricly speaking, this derivation is not correct for $n = 0$, although the answer is correct, since $\binom{-1}{k} = 0$ for $k > 0$, and $\binom{-1}{0} = 1$.

Compositons of n with k parts are really the same thing as n-element multisets from $[k]$ that use every element at least once. From our table of distribution problems, their number is the same as the number of $(n-k)$-element multisets from $[k]$, or $\binom{k+(n-k)-1}{n-k} = \binom{n-1}{n-k} = \binom{n-1}{k-1}$, in agreement with the answer above.

Exercise: modify the above example to allow compositions of n in which some of the parts can be zero. Note that as long as we use a two-variable generating function, counting the compositions by length as well as size, this does not put an element of weight zero in $\mathcal{A}$. The integer zero has weight 1 for the length, and is counted by the monomial y, so the modified $A(x,y)$ will still have no constant term.

3. Catalan numbers

We define the *Catalan number* C_n to be the number of words formed from n left and n right parentheses, with all parentheses balanced. We may tabulate the first few Catalan numbers by listing all the possibilities.

n	C_n	
0	1	$\emptyset$
1	1	()
2	2	()(), (())
3	5	()()(), (())(), ()(()), (()()), ((()))

Let

$$C(x) = \sum_{n=0}^{\infty} C_n x^n$$

be the generating function for Catalan numbers. We will use the sequence principle to evaluate it. Observe that each balanced parenthesis word, or *Catalan word*, can be uniquely expressed as a sequence of outermost groups

$$(\ w_1\ )(\ w_2\ )\cdots(\ w_k\ ),$$

where the words w_i inside each outer pair of parentheses are themselves Catalan words. Therefore, let us apply the sequence principle taking $\mathcal{A}$ to be the set of words of the form (w) with w a Catalan word. Note that the weight of (w) is one more than that of w, since the weight is the number of left parentheses, or equivalently, the number of right parentheses. In particular, there are no elements of weight zero in $\mathcal{A}$, even though w can be the empty word. We get each element of $\mathcal{A}$ by choosing a Catalan word, and surrounding it with an extra pair of parentheses. We can think of this extra pair as a forced "choice," with generating function x because it adds one to the weight. Thus by a trivial application of the product principle, we have the generating function

$$A(x) = xC(x).$$

However, by the sequence principle, we also have

$$C(x) = \frac{1}{1 - A(x)}.$$

Substiuting $xC(x)$ for $A(x)$ and clearing the denominator yields the quadratic equation

$$xC(x)^2 - C(x) + 1 = 0$$

for the Catalan number generating function. Hence

$$C(x) = \frac{1 - \sqrt{1 - 4x}}{2x}.$$

An interesting point is how we know whether to take $+\sqrt{1-4x}$ or $-\sqrt{1-4x}$ here. To see that the $-$ sign must be right, note that $2xC(x)$ must equal the numerator in the above fraction, so this numerator must have zero constant term as a formal series. Since $1 + \sqrt{1-4x}$ does not vanish at $x = 0$, it is the wrong numerator. Now we can use the extended binomial theorem to solve for the Catalan numbers C_n themselves. We have

$$2xC(x) = 1 - (1-4x)^{1/2} = 1 - \sum_{n=0}^{\infty} \binom{1/2}{n} (-4)^n x^n.$$

To extract C_n, compare coefficients of x^{n+1} on both sides, obtaining the rather messy formula

$$C_n = -\frac{(-4)^{n+1}}{2} \binom{1/2}{n+1} = (-1)^n 2^{2n-1} \binom{1/2}{n+1}.$$

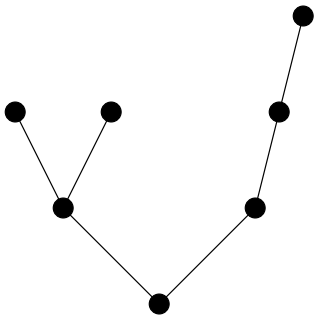
Writing out the binomial coefficient $\binom{1/2}{n+1} = (1/2)(-1/2)(-3/2)\cdots(-(2n-1)/2)/(n+1)!$, multiplying both numerator and denominator by $n!$, and simplifying, we eventually get

$$C_n = \frac{1}{n+1} \binom{2n}{n}.$$

There is another interpretation of Catalan numbers in terms of lattice paths, as follows. The condition for a parenthesis word to be balanced is that every initial segment of it have at least as many left parentheses as right parentheses. If we translate the word into a lattice path that takes one step north for each left parenthesis and one step east for each right parenthesis, then we get all paths from $(0, 0)$ to (n, n) that never go below the diagonal line $x = y$. Hence the Catalan number C_n is equal to the number of these special lattice paths (known as *Dyck paths*). There are many other interesting combinatorial objects whose number turns out to be C_n.

4. Ordered rooted trees

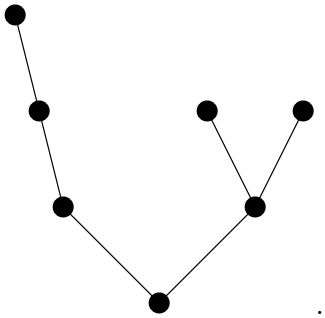
A *tree* is a connected graph without cycles. If we fix the set of vertices of our tree to be, say, $[n] = \{1, \ldots, n\}$, then we call the tree a *labelled* tree, the elements of the vertex set being the labels. It is a famous theorem of Cayley that there are n^{n-2} different labelled trees on n vertices. A *rooted* tree is a tree in which one of the vertices is distinguished and called the *root*. It is conventional to draw a rooted tree with the root at the bottom and the path from the root to each vertex going up, like this.



Since there are n choices for the root, there are n^{n-1} different rooted labelled trees on n vertices.

The vertex directly below a vertex v in a rooted tree is the *parent* of v (the root has no parent). The vertices directly above v, if any, are its *children*.

Besides labelled trees, we can also count various sorts of *unlabelled* trees. Technically, an unlabelled tree is an equivalence classes of labelled trees, two trees being equivalent if they differ only by a permutation of the labels. In this section we will only consider *ordered, rooted, unlabelled* trees. By *ordered*, we mean that in addition to the tree itself, there is given a linear ordering of the children of each vertex. In practice, the ordering is indicated implicitly by drawing the tree so that the children of each vertex are ordered left-to-right. This means, for instance, that the drawing above represents a different ordered tree than the tree



.

However, these two drawings would represent the same unordered, unlabelled tree.

We can use generating functions to count ordered rooted trees. Let t_n denote the number of these trees with n vertices, including the root, and let $T(x) = \sum_n t_n x^n$ be their generating function. To build any tree, we can first place the root, and then choose the (ordered) sequence of branches based on each child of the root. Each branch is itself an ordered rooted tree, and there can be any number of them. The generating function for the collection of branches is then

$$1/(1 - T(x))$$

by the sequence principle. We must multiply this by a factor of x to account for the additional vertex at the root, giving

$$T(x) = \frac{x}{1 - T(x)}.$$

As with the Catalan number generating function, this leads to a quadratic equation for $T(x)$:

$$T(x)^2 - T(x) + x = 0,$$

whose solution is

$$T(x) = \frac{1 - \sqrt{1 - 4x}}{2}.$$

Note that

$$T(x) = xC(x).$$

Hence, comparing coefficients of x^n, we see that $t_n = C_{n-1}$ for $n > 0$, while $t_0 = 0$, since by definition a rooted tree is not empty.

Incidentally, for ordered trees, it is easy to switch between the labelled and unlabelled versions of the problem. The ordering of the tree effectively "names" every vertex, so that on a labelled ordered rooted tree, every permutation of the labels gives a different tree. Hence the number of ordered, rooted labelled trees on n vertices is simply $n!t_n = n!C_{n-1}$.